

# Optimizing E-commerce Shipping Using Machine Learning

```
In [1]: import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.impute import SimpleImputer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler, MinMaxScaler
from sklearn.linear_model import LogisticRegression
from sklearn import tree
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report, roc_curve
```

## Importing Dataset

```
In [2]: data = pd.read_csv('Train1.csv')
```

## Data Description

- ID: ID Number of Customers.
- Warehouse block:- The Company have big Warehouse which is divided in to block such as A,B,C,D,E.
- Mode of shipment:- The Company Ships the products in multiple way such as Ship, Flight and Road.
- Customer care calls:- The number of calls made from enquiry for enquiry of the shipment.
- Customer rating:- The company has rated from every customer. 1 is the lowest (Worst), 5 is the highest (Best).
- Cost of the product:- Cost of the Product in US Dollars.
- Prior purchases:- The Number of Prior Purchase.
- Product importance:- The company has categorized the product in the various parameter such as low, medium, high.
- Gender:- Male and Female.
- Discount offered:- Discount offered on that specific product.
- Weight in gms:- It is the weight in grams.
- Reached on time:- It is the target variable, where 1 Indicates that the product has NOT reached on time and 0 indicates it has reached on time.

```
In [3]: data.head()
```

```
Out[3]:   ID  Warehouse_block  Mode_of_Shipment  Customer_care_calls  Customer_rating  Cost_of_the_Pro
```

0	1	D	Flight	4	2
1	2	F	Flight	4	5
2	3	A	Flight	2	2
3	4	B	Flight	3	3
4	5	C	Flight	2	2



```
In [4]: data.shape
```

```
Out[4]: (10999, 12)
```

```
In [5]: data.columns
```

```
Out[5]: Index(['ID', 'Warehouse_block', 'Mode_of_Shipment', 'Customer_care_calls',
       'Customer_rating', 'Cost_of_the_Product', 'Prior_purchases',
       'Product_importance', 'Gender', 'Discount_offered', 'Weight_in_gms',
       'Reached.on.Time_Y.N'],
      dtype='object')
```

```
In [6]: data.dtypes
```

```
Out[6]: ID                  int64
Warehouse_block        object
Mode_of_Shipment        object
Customer_care_calls     int64
Customer_rating         int64
Cost_of_the_Product    int64
Prior_purchases        int64
Product_importance      object
Gender                 object
Discount_offered        int64
Weight_in_gms           int64
Reached.on.Time_Y.N     int64
dtype: object
```

```
In [7]: data.isnull().sum()
```

```
Out[7]: ID                  0
Warehouse_block        0
Mode_of_Shipment        0
Customer_care_calls     0
Customer_rating         0
Cost_of_the_Product    0
Prior_purchases        0
Product_importance      0
Gender                 0
Discount_offered        0
Weight_in_gms           0
Reached.on.Time_Y.N     0
dtype: int64
```

```
In [8]: data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10999 entries, 0 to 10998
Data columns (total 12 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   ID               10999 non-null   int64  
 1   Warehouse_block  10999 non-null   object  
 2   Mode_of_Shipment 10999 non-null   object  
 3   Customer_care_calls 10999 non-null   int64  
 4   Customer_rating   10999 non-null   int64  
 5   Cost_of_the_Product 10999 non-null   int64  
 6   Prior_purchases   10999 non-null   int64  
 7   Product_importance 10999 non-null   object  
 8   Gender            10999 non-null   object  
 9   Discount_offered 10999 non-null   int64  
 10  Weight_in_gms    10999 non-null   int64  
 11  Reached.on.Time_Y.N 10999 non-null   int64  
dtypes: int64(8), object(4)
memory usage: 1.0+ MB
```

In [9]: `data.describe()`

Out[9]:

	ID	Customer_care_calls	Customer_rating	Cost_of_the_Product	Prior_purchases	D
<b>count</b>	10999.000000	10999.000000	10999.000000	10999.000000	10999.000000	
<b>mean</b>	5500.000000	4.054459	2.990545	210.196836	3.567597	
<b>std</b>	3175.28214	1.141490	1.413603	48.063272	1.522860	
<b>min</b>	1.000000	2.000000	1.000000	96.000000	2.000000	
<b>25%</b>	2750.50000	3.000000	2.000000	169.000000	3.000000	
<b>50%</b>	5500.00000	4.000000	3.000000	214.000000	3.000000	
<b>75%</b>	8249.50000	5.000000	4.000000	251.000000	4.000000	
<b>max</b>	10999.00000	7.000000	5.000000	310.000000	10.000000	

In [10]: `pd.DataFrame(data['Reached.on.Time_Y.N'].value_counts())`

Out[10]:

Reached.on.Time_Y.N	
1	6563
0	4436

## Preprocessing

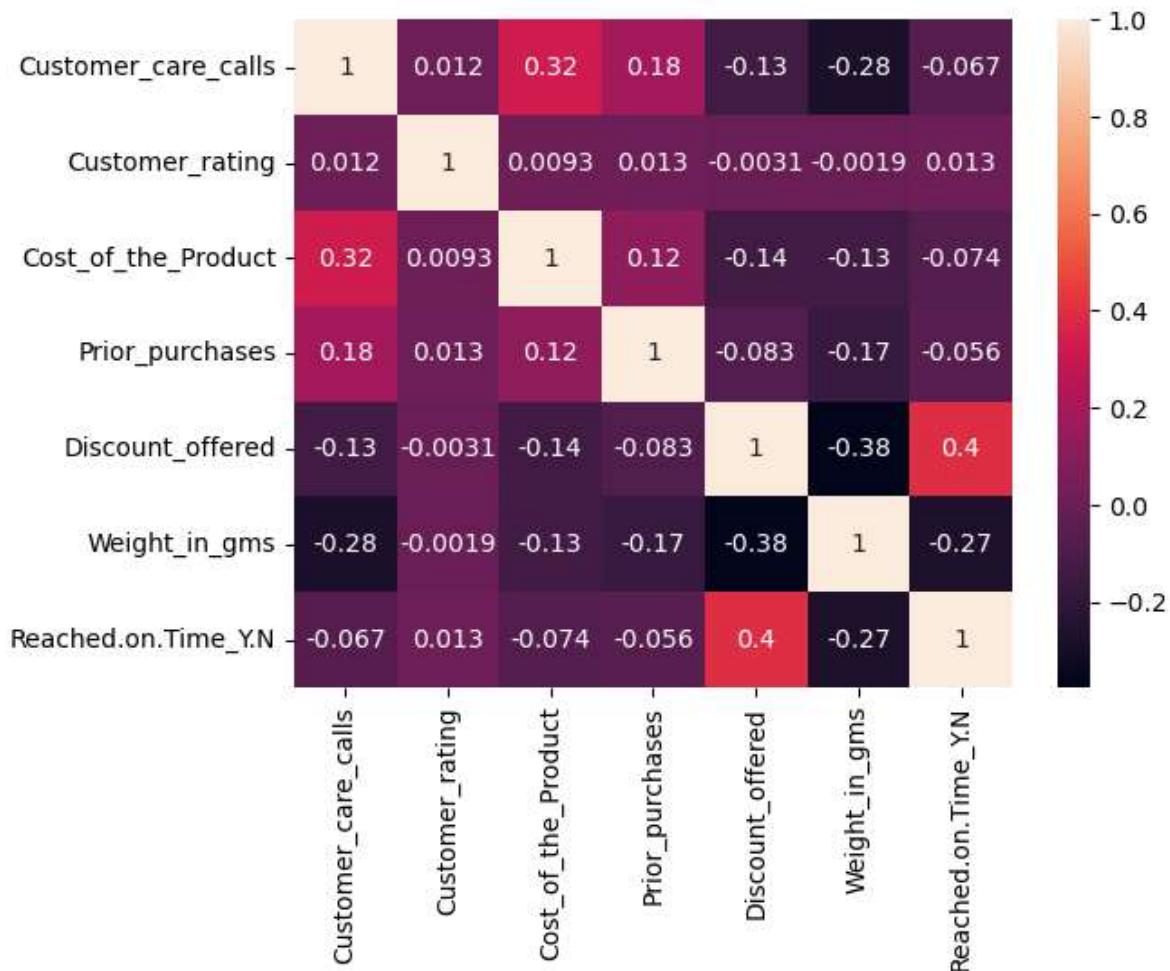
In [11]: `data = data.drop(['ID'], axis=1)`

In [12]: `cat_cols = data.select_dtypes('object')`  
`num_cols = data.select_dtypes('number')`  
`cat_cols.columns, num_cols.columns`

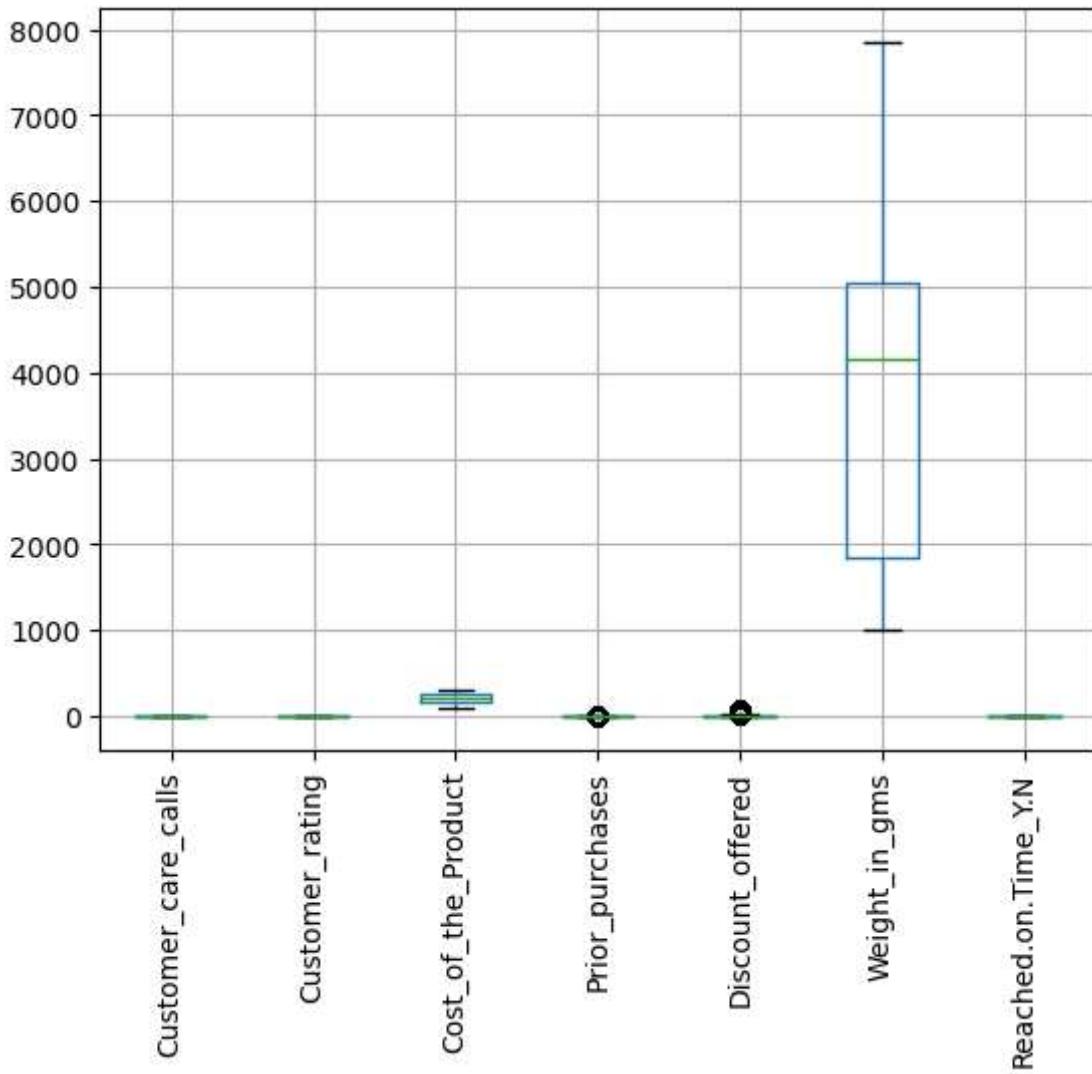
```
Out[12]: (Index(['Warehouse_block', 'Mode_of_Shipment', 'Product_importance', 'Gender'], dt  
ype='object'),  
Index(['Customer_care_calls', 'Customer_rating', 'Cost_of_the_Product',  
'Prior_purchases', 'Discount_offered', 'Weight_in_gms',  
'Reached.on.Time_Y.N'],  
dtype='object'))
```

```
In [13]: sns.heatmap(data.corr(), annot = True)
```

```
Out[13]: <AxesSubplot:>
```



```
In [14]: # outliers  
data.boxplot()  
plt.xticks(rotation=90)  
plt.show()
```



## Extract X and y

```
In [15]: X = data.drop(['Reached.on.Time_Y.N'], axis=1)
y = data['Reached.on.Time_Y.N']
```

```
In [16]: cat_cols = X.select_dtypes('object')
num_cols = X.select_dtypes('number')
```

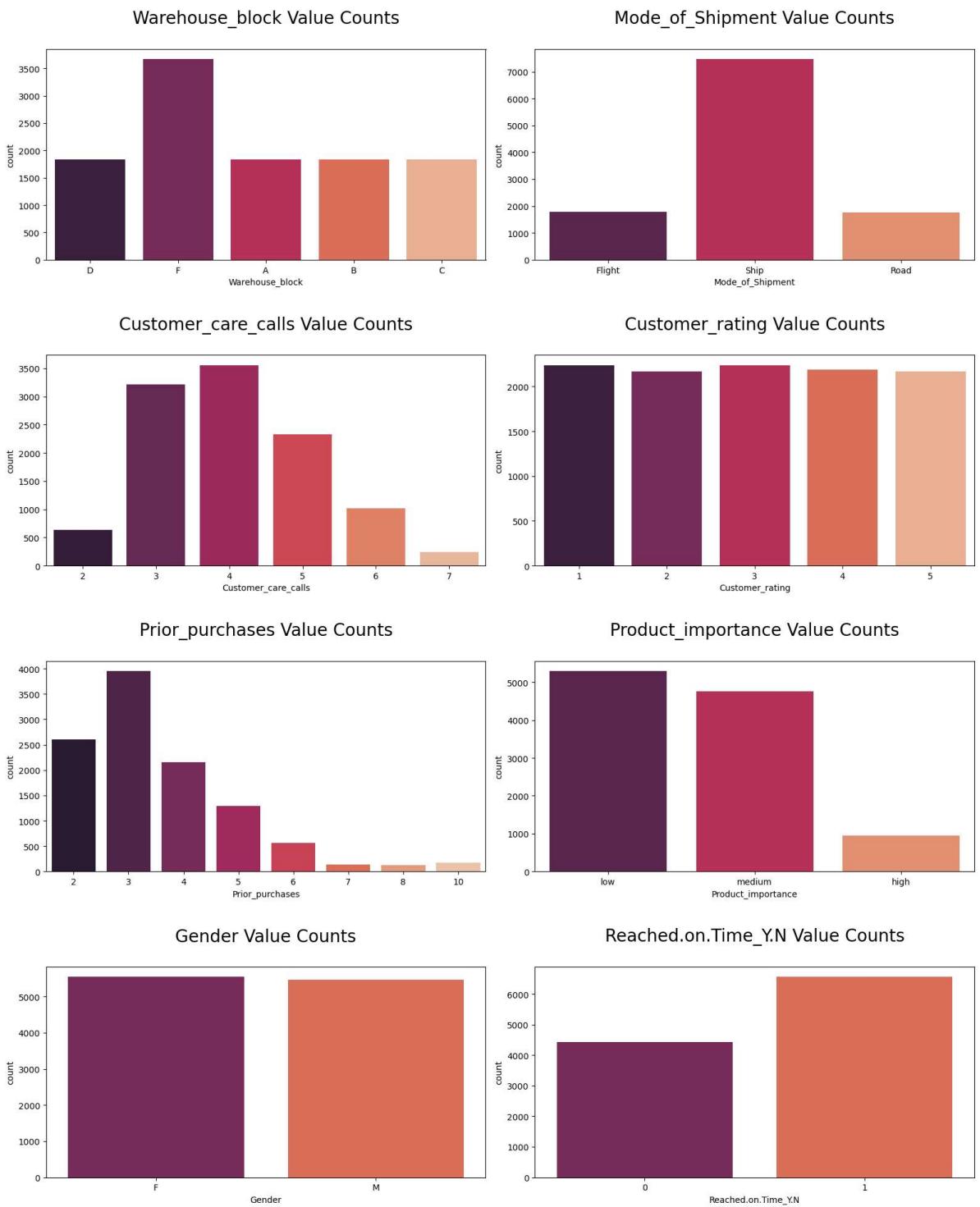
```
In [17]: cols = ['Warehouse_block', 'Mode_of_Shipment', 'Customer_care_calls', 'Customer_rating',
           'Prior_purchases', 'Product_importance', 'Gender', 'Reached.on.Time_Y.N']

plt.figure(figsize = (16, 20))
plotnumber = 1

for i in range(len(cols)):
    if plotnumber <= 8:
        ax = plt.subplot(4, 2, plotnumber)
        sns.countplot(x = cols[i], data = data, ax = ax, palette='rocket')
        plt.title(f"\n{cols[i]} Value Counts\n", fontsize = 20)

    plotnumber += 1

plt.tight_layout()
plt.show()
```

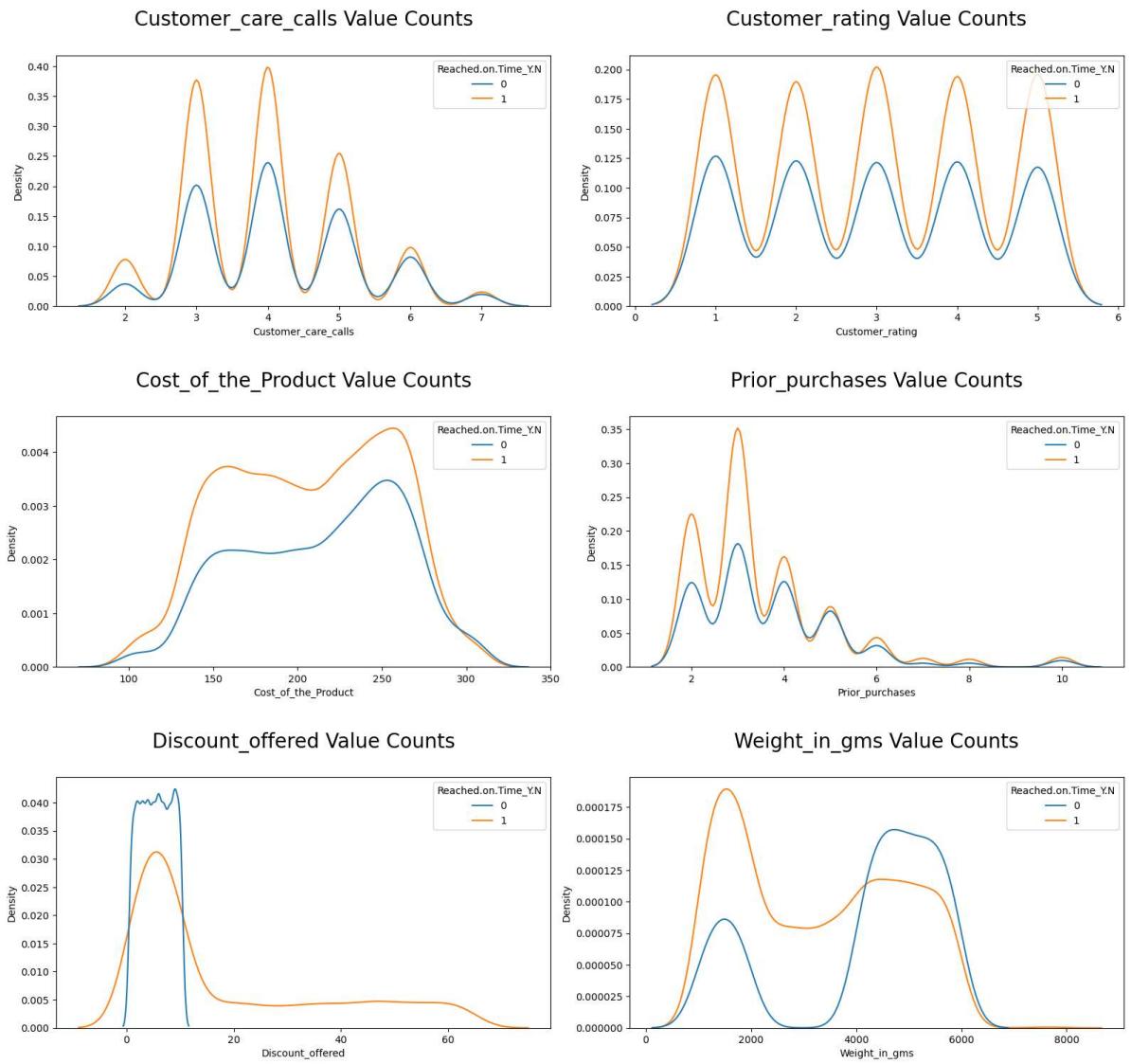


```
In [18]: cols = ['Customer_care_calls', 'Customer_rating', 'Cost_of_the_Product', 'Prior_pur
plt.figure(figsize = (16, 20))
plotnumber = 1

for i in range(len(cols)):
    if plotnumber <= 8:
        ax = plt.subplot(4, 2, plotnumber)
        sns.kdeplot(x = cols[i], data = data, ax = ax, hue='Reached.on.Time_Y.N')
        plt.title(f"\n{cols[i]} Value Counts\n", fontsize = 20)

    plotnumber += 1

plt.tight_layout()
plt.show()
```



```
In [19]: def encoder(df):
    for col in cat_cols:
        le=LabelEncoder()
        le.fit(df[col])
        df[col]=le.transform(df[col])
    return df
```

```
In [20]: X = encoder(X)
```

```
In [21]: X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.3, random_state=42)
```

```
In [22]: sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

## Existing solution and methodology

```
In [23]: classifier = LogisticRegression(solver='lbfgs' ,max_iter=9000)
classifier.fit(X_train,y_train)
y_pred = classifier.predict(X_test)
y_pred
```

```
Out[23]: array([1, 0, 0, ..., 1, 1, 1], dtype=int64)
```

```
In [24]: accuracy_score(y_test, y_pred)
```

```
Out[24]: 0.6339393939393939
```

```
In [25]: roc_auc_score(y_test,y_pred)
```

```
Out[25]: 0.6224412020905923
```

```
In [26]: print(classification_report(y_test,y_pred,zero_division=0))
```

	precision	recall	f1-score	support
0	0.54	0.57	0.55	1312
1	0.70	0.68	0.69	1988
accuracy			0.63	3300
macro avg	0.62	0.62	0.62	3300
weighted avg	0.64	0.63	0.64	3300

```
In [27]: rf = RandomForestClassifier(criterion= 'entropy', max_depth=4, min_samples_leaf= 3  
rf.fit(X_train, y_train)  
y_pred = rf.predict(X_test)  
y_pred
```

```
Out[27]: array([1, 0, 0, ..., 0, 1, 0], dtype=int64)
```

```
In [28]: accuracy_score(y_test, y_pred)
```

```
Out[28]: 0.6842424242424242
```

```
In [29]: roc_auc_score(y_test,y_pred)
```

```
Out[29]: 0.7311889630465721
```

```
In [30]: print(classification_report(y_test,y_pred,zero_division=0))
```

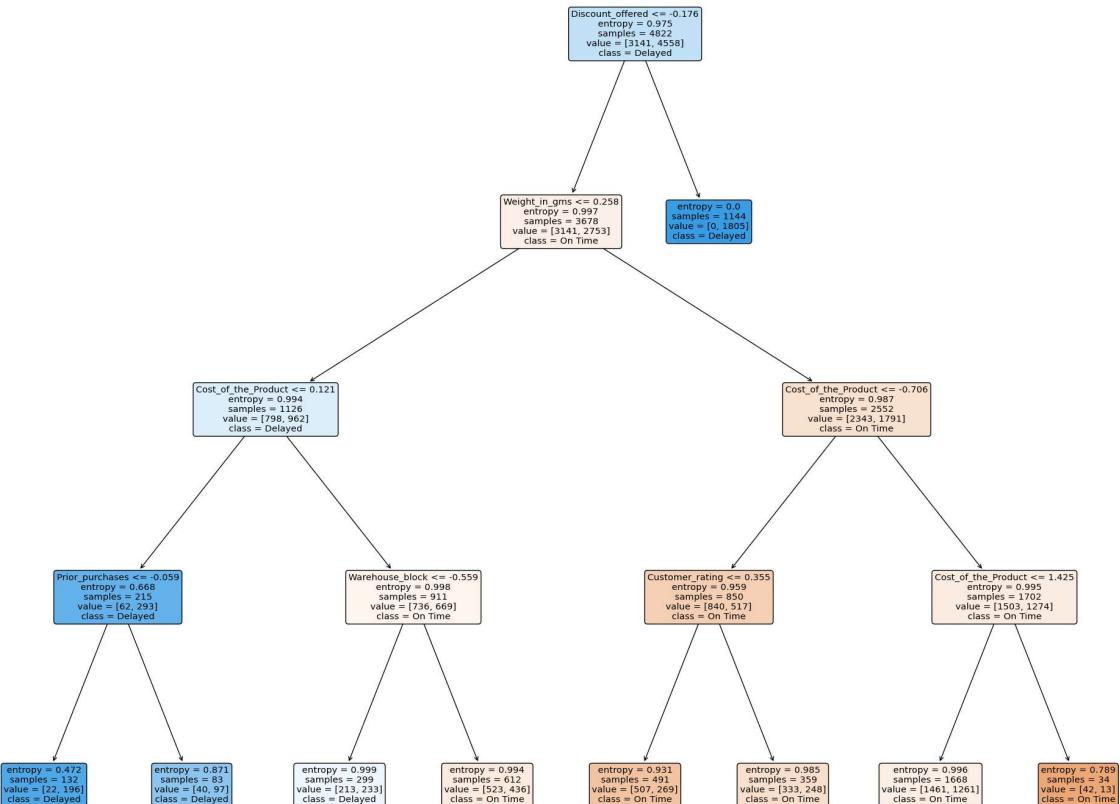
	precision	recall	f1-score	support
0	0.56	0.96	0.71	1312
1	0.95	0.50	0.66	1988
accuracy			0.68	3300
macro avg	0.76	0.73	0.68	3300
weighted avg	0.80	0.68	0.68	3300

```
In [31]: fig = plt.figure(figsize=(25,20))  
tree.plot_tree(rf.estimators_[1],feature_names=X.columns,class_names=['On Time', 'D
```

```

Out[31]: [Text(0.5625, 0.9, 'Discount_offered <= -0.176\nentropy = 0.975\nsamples = 4822\nvalue = [3141, 4558]\nclass = Delayed'),
Text(0.5, 0.7, 'Weight_in_gms <= 0.258\nentropy = 0.997\nsamples = 3678\nvalue = [3141, 2753]\nclass = On Time'),
Text(0.25, 0.5, 'Cost_of_the_Product <= 0.121\nentropy = 0.994\nsamples = 1126\nvalue = [798, 962]\nclass = Delayed'),
Text(0.125, 0.3, 'Prior_purchases <= -0.059\nentropy = 0.668\nsamples = 215\nvalue = [62, 293]\nclass = Delayed'),
Text(0.0625, 0.1, 'entropy = 0.472\nsamples = 132\nvalue = [22, 196]\nclass = Delayed'),
Text(0.1875, 0.1, 'entropy = 0.871\nsamples = 83\nvalue = [40, 97]\nclass = Delayed'),
Text(0.375, 0.3, 'Warehouse_block <= -0.559\nentropy = 0.998\nsamples = 911\nvalue = [736, 669]\nclass = On Time'),
Text(0.3125, 0.1, 'entropy = 0.999\nsamples = 299\nvalue = [213, 233]\nclass = Delayed'),
Text(0.4375, 0.1, 'entropy = 0.994\nsamples = 612\nvalue = [523, 436]\nclass = On Time'),
Text(0.75, 0.5, 'Cost_of_the_Product <= -0.706\nentropy = 0.987\nsamples = 2552\nvalue = [2343, 1791]\nclass = On Time'),
Text(0.625, 0.3, 'Customer_rating <= 0.355\nentropy = 0.959\nsamples = 850\nvalue = [840, 517]\nclass = On Time'),
Text(0.5625, 0.1, 'entropy = 0.931\nsamples = 491\nvalue = [507, 269]\nclass = On Time'),
Text(0.6875, 0.1, 'entropy = 0.985\nsamples = 359\nvalue = [333, 248]\nclass = On Time'),
Text(0.875, 0.3, 'Cost_of_the_Product <= 1.425\nentropy = 0.995\nsamples = 1702\nvalue = [1503, 1274]\nclass = On Time'),
Text(0.8125, 0.1, 'entropy = 0.996\nsamples = 1668\nvalue = [1461, 1261]\nclass = On Time'),
Text(0.9375, 0.1, 'entropy = 0.789\nsamples = 34\nvalue = [42, 13]\nclass = On Time'),
Text(0.625, 0.7, 'entropy = 0.0\nsamples = 1144\nvalue = [0, 1805]\nclass = Delayed')]

```



## Proposed Methodology

```
In [32]: dt = tree.DecisionTreeClassifier(criterion= 'entropy', max_depth=4, min_samples_leaf=1)
dt.fit(X_train, y_train)
y_pred = dt.predict(X_test)
y_pred
```

```
Out[32]: array([1, 0, 0, ..., 1, 1, 0], dtype=int64)
```

## Accuracy Score

```
In [33]: accuracy_score(y_test, y_pred)
```

```
Out[33]: 0.6772727272727272
```

```
In [34]: roc_auc_score(y_test,y_pred)
```

```
Out[34]: 0.7222941306374834
```

```
In [35]: print(classification_report(y_test,y_pred,zero_division=0))
```

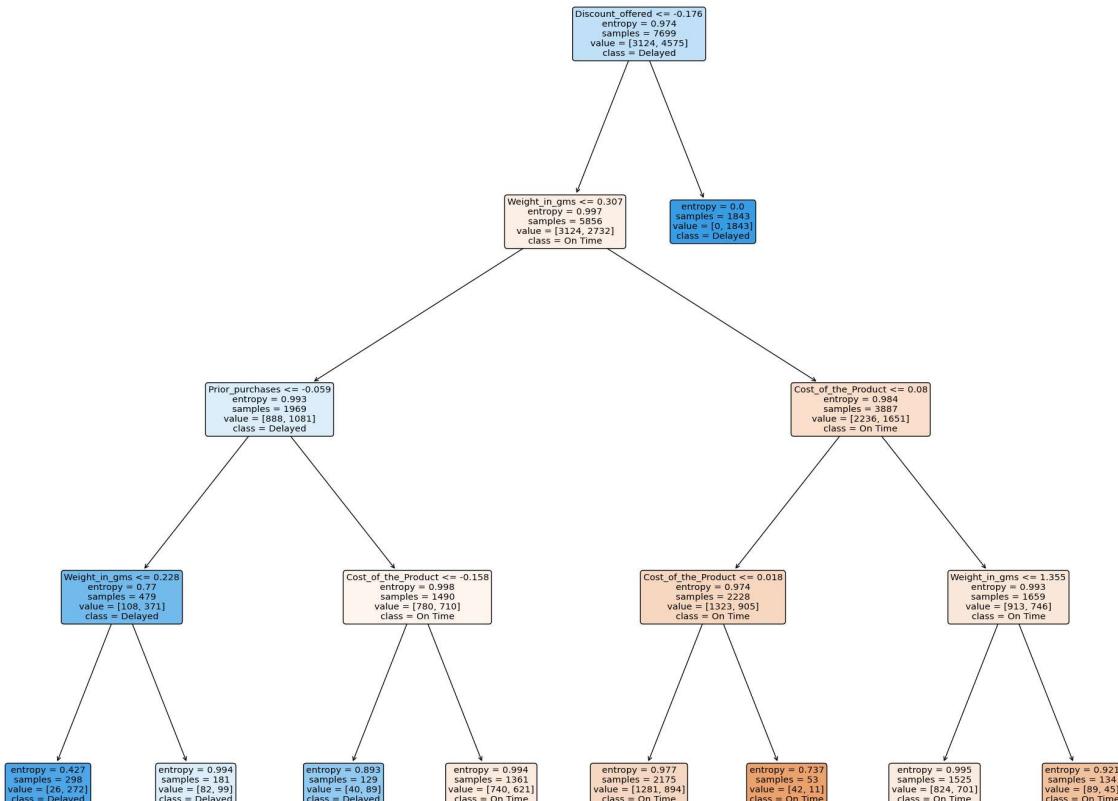
	precision	recall	f1-score	support
0	0.56	0.94	0.70	1312
1	0.93	0.50	0.65	1988
accuracy			0.68	3300
macro avg	0.74	0.72	0.68	3300
weighted avg	0.78	0.68	0.67	3300

```
In [36]: fig = plt.figure(figsize=(25,20))
tree.plot_tree(dt,feature_names=X.columns,class_names=['On Time','Delayed'],filled
```

```

Out[36]: [Text(0.5625, 0.9, 'Discount_offered <= -0.176\nentropy = 0.974\nsamples = 7699\nvalue = [3124, 4575]\nclass = Delayed'),
Text(0.5, 0.7, 'Weight_in_gms <= 0.307\nentropy = 0.997\nsamples = 5856\nvalue = [3124, 2732]\nclass = On Time'),
Text(0.25, 0.5, 'Prior_purchases <= -0.059\nentropy = 0.993\nsamples = 1969\nvalue = [888, 1081]\nclass = Delayed'),
Text(0.125, 0.3, 'Weight_in_gms <= 0.228\nentropy = 0.77\nsamples = 479\nvalue = [108, 371]\nclass = Delayed'),
Text(0.0625, 0.1, 'entropy = 0.427\nsamples = 298\nvalue = [26, 272]\nclass = Delayed'),
Text(0.1875, 0.1, 'entropy = 0.994\nsamples = 181\nvalue = [82, 99]\nclass = Delayed'),
Text(0.375, 0.3, 'Cost_of_the_Product <= -0.158\nentropy = 0.998\nsamples = 1490\nvalue = [780, 710]\nclass = On Time'),
Text(0.3125, 0.1, 'entropy = 0.893\nsamples = 129\nvalue = [40, 89]\nclass = Delayed'),
Text(0.4375, 0.1, 'entropy = 0.994\nsamples = 1361\nvalue = [740, 621]\nclass = On Time'),
Text(0.75, 0.5, 'Cost_of_the_Product <= 0.08\nentropy = 0.984\nsamples = 3887\nvalue = [2236, 1651]\nclass = On Time'),
Text(0.625, 0.3, 'Cost_of_the_Product <= 0.018\nentropy = 0.974\nsamples = 2228\nvalue = [1323, 905]\nclass = On Time'),
Text(0.5625, 0.1, 'entropy = 0.977\nsamples = 2175\nvalue = [1281, 894]\nclass = On Time'),
Text(0.6875, 0.1, 'entropy = 0.737\nsamples = 53\nvalue = [42, 11]\nclass = On Time'),
Text(0.875, 0.3, 'Weight_in_gms <= 1.355\nentropy = 0.993\nsamples = 1659\nvalue = [913, 746]\nclass = On Time'),
Text(0.8125, 0.1, 'entropy = 0.995\nsamples = 1525\nvalue = [824, 701]\nclass = On Time'),
Text(0.9375, 0.1, 'entropy = 0.921\nsamples = 134\nvalue = [89, 45]\nclass = On Time'),
Text(0.625, 0.7, 'entropy = 0.0\nsamples = 1843\nvalue = [0, 1843]\nclass = Delayed')]

```



```
In [37]: confusion_matrix(y_test,y_pred)
```

```
Out[37]: array([[1236,    76],  
   [ 989,  999]], dtype=int64)
```

```
In [38]: print(classification_report(y_test,y_pred,zero_division=0))
```

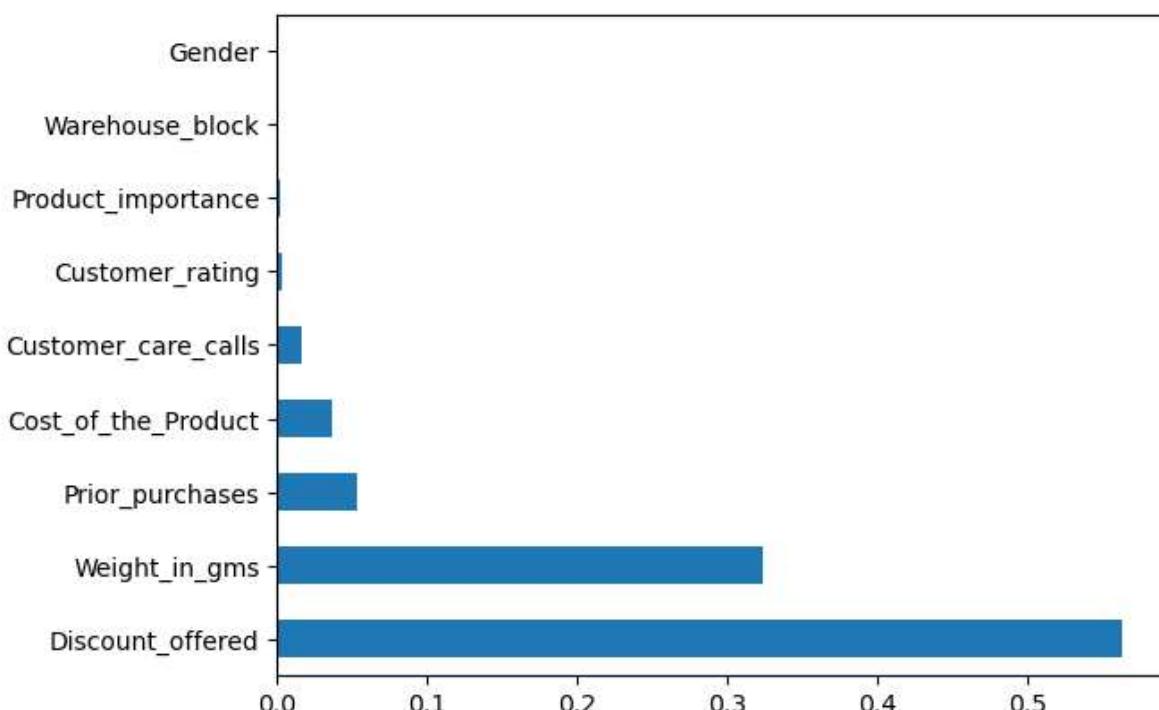
	precision	recall	f1-score	support
0	0.56	0.94	0.70	1312
1	0.93	0.50	0.65	1988
accuracy			0.68	3300
macro avg	0.74	0.72	0.68	3300
weighted avg	0.78	0.68	0.67	3300

```
In [42]: feat_importances = pd.Series(rf.feature_importances_, index=X.columns)  
feat_importances
```

```
Out[42]: Warehouse_block      0.001173  
Mode_of_Shipment           0.000592  
Customer_care_calls        0.016427  
Customer_rating             0.003085  
Cost_of_the_Product         0.036800  
Prior_purchases            0.053027  
Product_importance          0.002088  
Gender                      0.000793  
Discount_offered            0.562633  
Weight_in_gms               0.323381  
dtype: float64
```

```
In [44]: feat_importances.nlargest(9).plot(kind='barh')
```

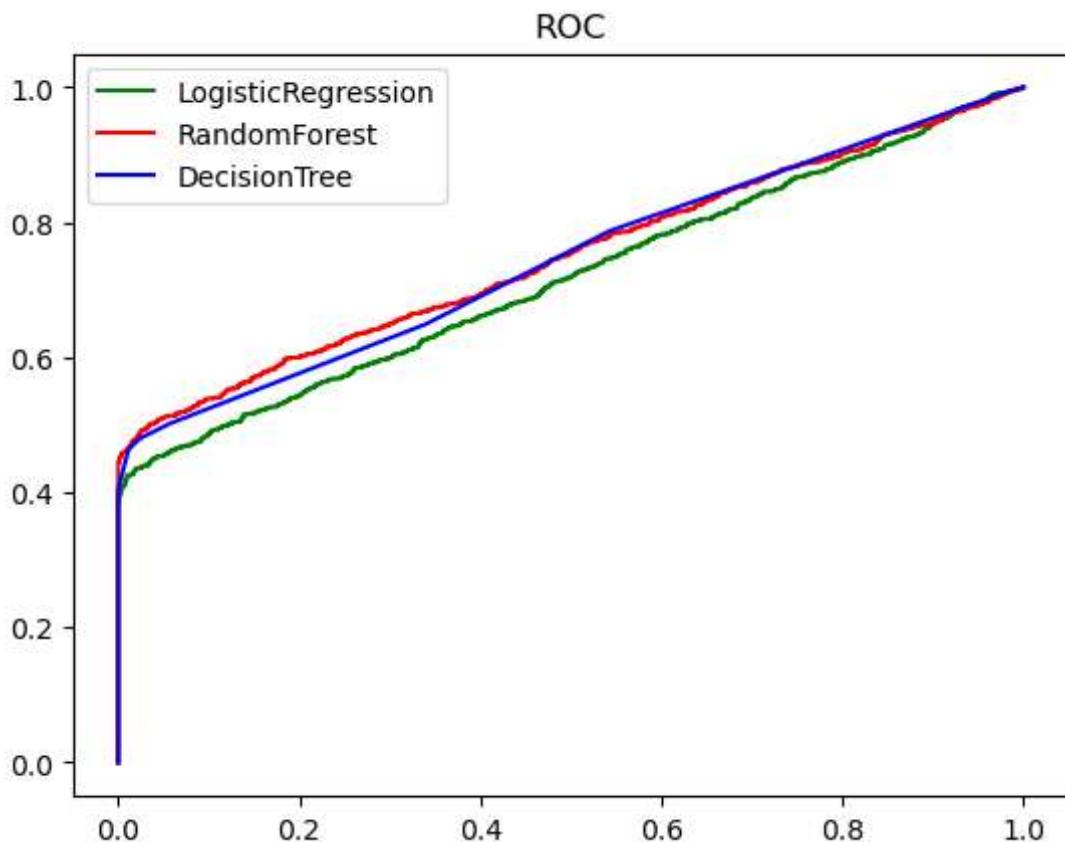
```
Out[44]: <AxesSubplot:>
```



## ROC AUC Curve

```
In [45]: pred_prob = classifier.predict_proba(X_test)
pred_prob1 = rf.predict_proba(X_test)
pred_prob2 = dt.predict_proba(X_test)
fpr,tbr,threshold = roc_curve(y_test, pred_prob[:,1],pos_label=1)
fpr1,tbr1,threshold1 = roc_curve(y_test, pred_prob1[:,1],pos_label=1)
fpr2,tbr2,threshold2 = roc_curve(y_test, pred_prob2[:,1],pos_label=1)
plt.plot(fpr, tbr, linestyle='--', color='green', label='LogisticRegression')
plt.plot(fpr1, tbr1, linestyle='--', color='red', label='RandomForest')
plt.plot(fpr2, tbr2, linestyle='--', color='blue', label='DecisionTree')

plt.title('ROC')
plt.legend(loc = 'best')
plt.show()
```



```
In [ ]:
```